ADVANCED INTELLIGENT SYSTEMS
Open Access
www.advintellsyst.com

Onboard Evolution of Understandable Swarm Behaviors

Simon Jones, Alan F. Winfield, Sabine Hauert, and Matthew Studley**

Designing the individual robot rules that give rise to desired emergent swarm behaviors is difficult. The common method of running evolutionary algorithms off-line to automatically discover controllers in simulation suffers from two disadvantages: the generation of controllers is not situated in the swarm and so cannot be performed in the wild, and the evolved controllers are often opaque and hard to understand. A swarm of robots with considerable on-board processing power is used to move the evolutionary process into the swarm, providing a potential route to continuously generating swarm behaviors adapted to the environments and tasks at hand. By making the evolved controllers human-understandable using behavior trees, the controllers can be queried, explained, and even improved by a human user. A swarm system capable of evolving and executing fit controllers entirely onboard physical robots in less than 15 min is demonstrated. One of the evolved controllers is then analyzed to explain its functionality. With the insights gained, a significant performance improvement in the evolved controller is engineered.

1. Introduction

Swarm robotics takes inspiration from collective phenomena in nature,[1] where swarm-level behaviors emerge through the local interactions of multiple agents with each other and with the environment. Swarms have appealing properties for robotic systems; they are robust, resilient, and scalable and show potential in real-world applications ranging from exploration, mapping, and search and rescue to disaster recovery, pollution control, and cleaning.[2] A central problem within the field is the design of controllers for the individual agents such that the desired swarm behavior emerges. Artificial evolution has been widely used to allow swarm engineers to automatically discover controllers capable of producing the desired collective behavior.[3]

Conventionally, evolutionary swarm robotics has used two approaches: off-line evolution in simulation, followed by the transfer of the evolved controllers into a real swarm, and online embodied evolution within the swarm, where robots continually test the fitness of controllers in the real world and exchange genetic material to generate new controllers. Off-line evolution requires external infrastructure to perform the evolution and send the resulting controllers to the robots. It also requires good a priori information about the environments and scenarios the robots may encounter, which often results in a reality gap when a mismatch is present, causing evolved controllers to perform poorly in reality. Online embodied evolution can be slow, taking hours or days, and has the danger that a robot instantiating a bad controller could come to harm, an important consideration in potentially hostile environments. We propose a hybrid, moving computation into the real swarm so that evolution can be both fast, as with off-line evolution, and potentially responsive to changes in the environment, as embodied evolution is.

Both off-line and online evolutionary methods have typically resulted in controllers that are opaque and hard to understand. This has important implications for safety analysis and the ability to gain insight from the discovered controllers and even improve them. We use behavior trees (BT) as the controller architecture. They have desirable properties, they are hierarchical, so any subtree is a valid behavior tree in its own right, they are modular and can be used to encapsulate useful sub-behaviors, and they are human-readable and amenable to automatic simplification.

In this article, we outline our first steps toward the vision of an adaptive, responsive, and safe swarm by using on-board evolution in simulation with the Xpuck Teraflop swarm, enhanced e-pucks with very high collective processing power,[4] followed by analysis, understanding, and improvement of one of the evolved behavior trees. We make two contributions, firstly, we demonstrate in-swarm evolution of controllers in real time that result in the real-world swarm fitness improving over time, with very fit controllers emerging in some cases in less than 15 min. And secondly, we demonstrate a benefit of behavior trees as a controller architecture by showing how it is possible to simplify, analyze, and then improve one of the evolved controllers.

This article is arranged in the following way, Section 2 discusses the background and situates the study. Section 3 describes the methods, and in Section 4, we present the results and discuss

S. Jones, Dr. S. Hauert
Bristol Robotics Laboratory
University of Bristol
Senate House, Tyndall Ave, Bristol BS8 1TH, UK
E-mail: sabine.hauert@bristol.ac.uk

Prof. A. F. Winfield, Dr. M. Studley
Bristol Robotics Laboratory
University of the West of England
Coldharbour Ln, Stoke Gifford, Bristol BS16 1QY, UK
E-mail: matt.studley@brl.ac.uk

The ORCID identification number(s) for the author(s) of this article can be found under https://doi.org/10.1002/aisy.201900031.

DOI: 10.1002/aisy.201900031

them. Finally in Section 5, we draw some conclusions and outline further work.

2. Background and Previous Work

Common approaches to engineering swarm behaviors include bioinspiration, evolution, reverse engineering, and hand design.[5–10] Controller architectures include neural networks, probabilistic finite state machines (FSM), behavior trees, and hybrid combinations.[11–14] See Francesca and Birattari for a recent review.[15]

We use BT for our controller architecture because they are modular, human readable, and extendable. A BT is a hierarchical structure of nodes with leaves that interact with the world and inner nodes that combine these actions in various ways. All FSMs can be represented by a BT and with a source of randomness so can probabilistic FSMs.[16] The tree structure of BTs is amenable to the techniques of Genetic Programming.[17] They have their origins as a software engineering tool but are now widely used in the games industry as the controllers of non-player characters. Recent studies have formalized and applied them to robotics,[18–20] with our previous study applying them to evolutionary swarm robotics.[14]

When controllers are discovered through evolution or other automatic methods using a simulated environment, the problem of transferability of the controller to real robots arises, the so-called reality gap. Approaches to minimizing this include using high-fidelity simulation with periodic testing on real robots,[21,22] injection of noise within a simulation,[23] including transferability within the fitness function of the automatic method,[24,25] and reducing the representational power of the controller.[12] We apply a combination of techniques, injecting noise, minimizing the effect of problematic areas of simulation such as collisions by avoiding behaviors that give rise to them, and using the ability of behavior trees to encapsulate predesigned useful sub-behaviors.

Embodied evolution in robotics directly tests controllers in reality, avoiding the reality gap problem. When applied to swarms, the evolutionary algorithm is distributed over the robots achieving parallelism with individual agents testing different controllers and robots "mating" to generate controller solutions.[26–29] The use of real robots to evaluate the controllers means run times can be very long, days or even weeks. See Bredeche et al. for a recent review.[30]

The robot platform we use is our Xpuck Teraflop swarm.[4] Based on the e-puck, it extends its computational capabilities using a powerful single-board computer with in excess of 130 GFLOP graphics processing unit (GPU)-based processing performance.[31] The nine-robot swarm we use in this study has a collective processing power of over 1 teraflop. We distribute evolution of behavior tree controllers over the swarm, where each Xpuck is a node within a distributed parallel island model evolutionary algorithm (EA). There is a large literature on parallel EAs.[32] The island model separates the population into islands that evolve along separate trajectories but between which there is a certain level of migration of individuals. The island model often yields better results than a single panmictic population of the same size, due to diversity being maintained.[33] Topology, migration frequency, and migration size are important parameters defining the properties. O'Dowd et al. use an island model system within a swarm of e-pucks enhanced with the Linux Extension Board;[34,35] however, they only simulate a single robot. Each of our robots runs several hundred parallel simulations of the whole swarm.

Because of the advances in processing power available to build computationally powerful swarms and the explainability offered by behavior trees, this is the first study to combine fast on-board evolution of swarm robotic behaviors and the understandability of behavior trees. The time is ripe to move swarms into the real world.

3. Experimental Section

3.1. Benchmark Task and Fitness Function

A benchmark task is needed for the swarm of robots that is non-trivial and has relevance to possible real-world applications. In the field, foraging is regarded as canonical problem in that it encapsulates the solution of many sub-problems, such as navigation, object recognition, and transport, and that it is a direct analogue to real-world problems, such as harvesting, pollution control, search and rescue, and many others.[36]

The version used in this study required the swarm to continuously move a stream of objects in a particular direction. The direct manipulation of real objects was important, to demonstrate resilience to reality gap effects, so a round blue plastic Frisbee was used. The Xpuck robots had no manipulators, so the frisbee could only be moved with pushing actions. All experiments took place in an arena of size 2 m by 1.5 m surrounded by white walls of height 0.2 m. In the arena were placed nine Xpuck robots and the blue plastic frisbee. All objects within the arena were tracked with a Vicon motion tracking system by means of unique patterns of spherical reflectors. The robots were connected by Wi-Fi to a Hub PC that was used to initiate experiments and captured data, illustrated in **Figure 1**.

The task was defined as follows: The blue frisbee was placed approximately in the center of the arena. The swarm must move the frisbee in the $-x$ direction. If the frisbee contacted either the $+x$ or the $-x$ walls of the arena, the robots were stopped and the frisbee relocated back to the approximate center before the robots were started again at their current location. The fitness of the swarm was the $-x$ velocity of the frisbee normalized to the maximum Xpuck velocity and averaged over some time period.

The fitness function is then

$$f_{\text{raw}} = -\frac{\sum \Delta x_{\text{frisbee}}}{t_{\text{sim}} \cdot v_{\max}} \tag{1}$$

$$k_{\text{penalty}} = \begin{cases} 1 & \text{if } \Sigma \Delta x_{\text{frisbee}} = 0 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

$$r_{\text{derate}} = \begin{cases} 2 \cdot r_{\text{memfree}} & \text{if } r_{\text{memfree}} < 0.5 \\ 1 & \text{otherwise} \end{cases} \tag{3}$$

$$f_{\text{evo}} = r_{\text{derate}} \cdot (f_{\text{raw}} - k_{\text{penalty}}) \tag{4}$$

where $\Delta x_{\text{frisbee}}$ are the movements of the frisbee in the x direction, not counting relocations. For use within the evolutionary algorithm, the raw fitness value was modified in two ways. First, the case where there was no movement of the frisbee at

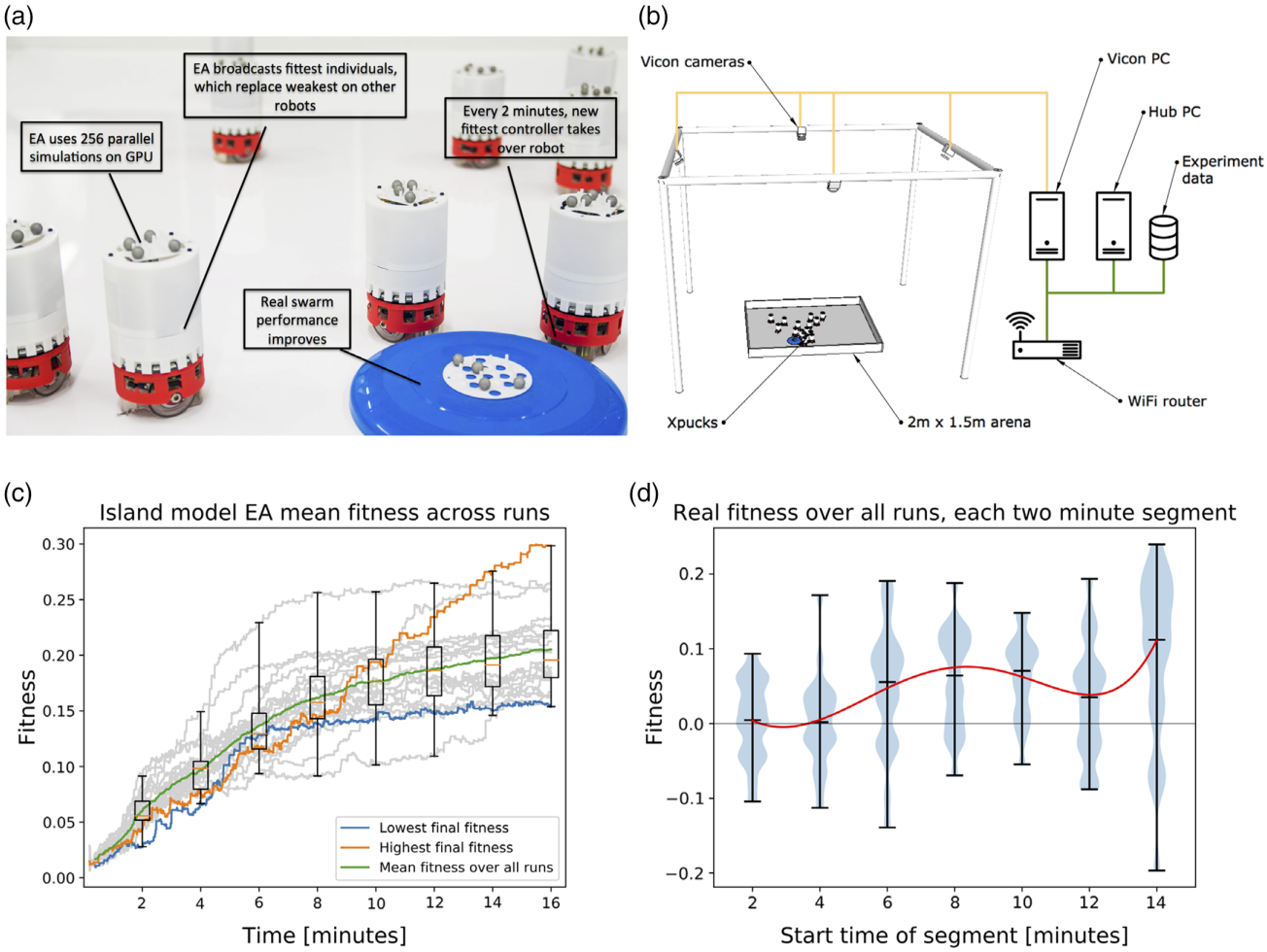

Figure 1. a) Image of the swarm running, showing several Xpucks, with two pushing the Frisbee. b) The Xpuck arena. Experiments take place within a 2 m × 1.5 m area surrounded by walls slightly higher than the height of the Xpucks. Each Xpuck has a unique pattern of spherical reflectors on their top surface to enable the Vicon motion tracking system to identify each individual object pose. The Vicon PC is dedicated to managing the Vicon system and makes available a stream of pose data. The Hub PC is responsible for all experiment management, data logging, and virtual sense synthesis. c) Mean fitness of the island model EA over each of the 20 runs. Boxplot whiskers cover full range. d) Real fitness of the swarm over time across all runs. Violin plots show distribution of fitness over runs, with ticks at median and extrema. Red line is fifth-order polynomial fitted to medians of each segment.

all was penalized to bootstrap into solutions where the robots were at least moving; a randomly moving robot that collided with the frisbee was better than non-moving robots. Second, the fitness was derated when the amount of available memory r_{memfree} fell below 50% to control tree bloat.

3.2. Xpuck Reference Model

The Xpuck robots were based on the e-puck and used the same physical sensors, with the addition of substantial processing power to enable the fast on-board physics-based simulation. The various sensor and actuator capabilities needed to be both modeled in simulation and exposed to the robot controller. This was formalized with the robot reference model, shown in **Table 1**. This approach and the design of some behavior tree functionality were inspired by Francesca et al.[37] The robot was based on the e-puck and used the same physical sensors and actuators. It was a two-wheel differential drive robot with a maximum speed of 0.13 ms^{-1}. There were eight IR proximity sensors around its perimeter, capable of sensing an obstacle out to a few centimeters away, at a height about 35 mm above the ground, capable of detecting other Xpucks and the arena walls, but not the frisbee, which only had a height of about 20 mm. A VGA camera together with image processing code could detect blobs of color. Only blue detection was used in this article to see the blue frisbee used in the benchmark task. The image processing produced a three bit number, indicating the presence of blue in the left, center, or right thirds of the field of view of the camera.

In addition, the robot was augmented with two virtual senses, using the pose information available from the Vicon system in the arena. These were a compass, giving the pose angle of a robot in the world frame, and a range-and-bearing sense, giving the number of neighbors and each of their range and bearings, out to a maximum range of half a meter. Both of these senses could be implemented on the real robots with additional hardware.

Table 1. Robot reference model for the Xpucks.

Input variables	Values	Description
$P_{i\in\{1,2,..,8\}}$	[0,1]	Proximity sensor i
$B_{i\in\{\text{left,center,right}\}}$	{0,1}	Blue blob detection
Θ	$[-\pi,\pi\}$	Compass, giving pose angle in world frame
$n\in\mathbb{N}$	{0,...,15}	Number of neighboring Xpucks
$(r,\angle b)_{i\in\{1,\ldots,n\},n\neq 0}$	$([r_{\min},r_{\max}],[-\pi,\pi))$	Range and bearing of neighbor i
Output variables		
$v_{i\in\{\text{left,right}\}}$	$[-v_{\max},v_{\max}]$	Left and right wheel velocities
Constants		
t_{update}	100 ms	Sensor and controller update period
$r_{\min}$	75 mm	Minimum range and bearing range
$r_{\max}$	0.5 m	Maximum range and bearing range
$v_{\max}$	0.13 ms^{-1}	Maximum wheel velocity
$\angle q_{i\in\{1,2,\ldots,8\}}$	17°, 49°, 90°, 150°, −150°, −90°, −49°, −17°	Angle of proximity sensor i
$p_{\max}$	30 mm	Proximity sensor maximum range
p_{height}	35 mm	Height of proximity sensors above ground
FOV	56°	Camera field of view
d	75 mm	Diameter of robot
l	53 mm	Wheelbase

3.3. Behavior Tree Architecture

A behavior tree controller architecture was used. Behavior trees had desirable properties such as modularity, encapsulation, and human readability. Any behavior tree could be used as a subtree within another; any subtree is a valid behavior tree in its own right.

The tree structure meant that they were amenable to the tree crossover and mutation techniques from genetic programming.[17] Terminology varied slightly across the literature, but all behavior trees had a set of nodes in a tree structure. All nodes and thus subtrees had the same interface; they received tick events from their parent and responded immediately with one and only one of *success* ≡ *S*, *failure* ≡ *F*, or *running* ≡ *R* if the subtree was performing some task that took non-zero time. The root of the tree was the source of regular tick events, usually at the robot controller update rate. Inner nodes received ticks and responded based on the responses of their children to ticks. Leaf nodes interacted with the environment, represented in abstraction as the blackboard, a set of registers that could be read and written.

The inner nodes were common across different implementations of behavior trees, the leaf nodes and blackboard were domain-specific and designed for the particular application. The most important inner nodes were the sequence and selection nodes, abbreviated as `seq` and `sel`. These combined at least two child subtrees in complimentary logical ways: `seq` ticked each of its subtrees in left-to-right order until they have all returned success or any return failure or running, returning that respectively, and `sel` ticked each subtree until they have all returned failure or any return success or running. Additional inner nodes termed decorators had a single-child subtree and performed operations like logical inversion and repetition.

The leaf nodes and the blackboard defined the interface between the controller and the real world. Using the robot capabilities described by the robot reference model, a set of blackboard entries and nodes to manipulate them was defined. To move the robot, $\boldsymbol{v}_{\mathbf{goal}}$ specified a target direction vector for motion. The senses were expressed as $\boldsymbol{v}_{\mathbf{blue}}$, which pointed toward any blue objects visible in the forward facing camera, $\boldsymbol{v}_{\mathbf{up}}$ pointing in the $+x$ direction, $\boldsymbol{v}_{\mathbf{attr}}$ pointing toward large concentrations of other robots, and $\boldsymbol{v}_{\mathbf{prox}}$, which pointed to the nearest obstacle detected by the IR sensors. Leaf nodes provided ways of manipulating blackboard entries, providing for scaling, rotation, and addition. Blackboard entries could be queried in various ways, including probabilistically. Please see Supporting Information S1 for more detailed information.

3.4. Automatic Tree Reduction

To improve understandability of the trees, an automatic method of tree reduction, akin to compiler optimization, was formalized. A series of reduction transformation rules that could be applied to a tree while leaving its functionality unchanged were specified (see Supporting Information S2 for more detailed information). An example would be that any subtree of a `seq` known to return failure meant that subsequent child subtrees to its right would never be ticked and could be removed.

Since the reduction rules were identities, the execution of a correctly reduced tree must result in identical behavior, anything else indicating bugs in the process. To validate our reductions, a simulation was run with nine robots executing the original and reduced tree for 60 simulated seconds, in each case producing a log file containing the poses of all objects at every timestep, together with all sensor inputs and actuator outputs. Any difference in the log files indicated non-equivalence.

3.5. Simulator and Reality Gap Mitigation

The simulator used in this study was detailed in Jones et al.[4] It was a fast 2D physics simulator and behavior tree interpreter that ran on the GPU of the Xpuck. In order that the simulator could be used successfully to evolve controllers that transfer well to the real robots, the effect of the reality gap must be minimized. There is always a trade-off between higher simulator fidelity and faster simulation, so fidelity must be improved where the simulation performance cost is low, while using other mitigating strategies of noise injection and behavior modification.

Three approaches were used. First, a series of simple scenarios were run with one or two robots pushing the frisbee in the real arena. Using the captured pose data from the Vicon system, these scenarios were recreated in simulation and the simulator parameters associated with friction and collisions were tuned

such that the differences between simulator and real trajectories were minimized. The latencies of camera color detection, synthesized range-and-bearing, and compass senses were also measured, and it was ensured that the simulator matched these.

Second, the repeatability of motion was measured to get real-world information about the noise of the robots. A much higher level (10×) of motion noise was then injected into the simulator to mask simulator infidelities.[23]

Finally, observing that collisions between robots and between robots and walls are the most problematic and expensive from a modeling perspective, and the area of largest discrepancy between trial scenarios and simulation, collisions at the controller level were minimized. All robots ran a base-level collision avoidance behavior; if any obstacle was sensed by the IR proximity detectors, the robot would turn away from the obstacle. Due to the modular hierarchical nature of behavior trees, this was simple to implement as a top-level tree, with the evolved controller tree being instantiated below this.

3.6. In-Swarm Evolution

In order that in-swarm evolution could be performed, an evolutionary algorithm capable of running across the multiple robots of the swarm was needed. To do this, an evolutionary algorithm was run on each robot, and they were connected by migrating individuals between robots. This is the island model distributed evolutionary algorithm. It took inspiration from the way that natural evolution proceeds on islands. Each island hosted a population of evolving individuals with its own evolutionary trajectory. In addition, there was some degree of interchange of genetic material between the island, a migration rate. The separation into subpopulations could result in higher performance than a single panmictic population of the same size due to niching effects and the maintenance of diversity.[38] In addition, by separating the total population into sub-populations with only a small amount of communication between then, coarse-grained parallelism was enabled.

It is generally the case that robot simulation time scales with the number of robots being simulated.[4] As robots were added to the swarm, the collective processing power increased, compensating for the required additional processing required to simulate that larger swarm such that simulation time and thus the evolutionary algorithm generation time remained approximately constant. The swarm was scalable in evolutionary performance.

Evolution proceeds in the following way using the fitness function detailed earlier. On each robot, a population $n_{\text{pop}} = 256$ of new individuals was generated using Koza's ramped-half-and-half procedure with a maximum tree depth of $n_{\text{depth}} = 6$.[39] The fitness of the population was measured in simulation over $t_{\text{sim}} = 60$ s with a single evaluation, and then sorted. A new population of individuals was formed from this population; the fittest $n_{\text{elite}} = 64$ individuals were transferred across unchanged. The remaining individuals were either copied across unchanged or with probability $p_{\text{replace}} = 0.25$ replaced with either a new random individual or an individual generated by crossover using a modified tournament selector from two elite parents with probability $p_{\text{xover}} = 0.5$, followed by the three mutation operators with probabilities for parameters of $p_{\text{mutparam}} = 0.05$, node replacement $p_{\text{mutpoint}} = 0.05$, or new subtree $p_{\text{mutsubtree}} = 0.05$. The fitness of the new population was measured in the same way as previously. Because many (81%) of individuals were unchanged, some would undergo multiple fitness evaluations, providing resilience to the noisy fitness function. The algorithm maintained statistics on the number of evaluations, the average fitness, and the variance.

The modified tournament selector used a size of $n_{\text{tsize}} = 3$ but instead of comparing average fitness of the selected individuals, it compared the 95% likelihood fitness, or if there has been only one evaluation, half the fitness. This exerted some selection pressure toward lower variance in fitness.

After each new generation was completed on a particular robot, the fittest individual of that population was broadcasted. All robots in the swarm sample made a copy of currently broadcast individuals at the point they started broadcasting their own fit individual. They maintained a list of the eight most recent sampled individuals from each robot. The least fit eight individuals of the local population were replaced by the fittest eight individuals across these lists of recent fit individuals from other robots. This gave the island model migration rate of $r_{\text{migration}} = \frac{8}{256} = 0.031$. This process was asynchronous and decentralized; robots would not finish each generation in step. The migration topology was fully connected because all robots could hear the broadcasts of any other.

Every 2 min, the behavior tree execution engine of each Xpuck loaded the latest, fittest controller that the local evolutionary algorithm had generated. This controller took over the running of the robot in the real world from that point until the next controller was loaded. The real swarm thus executed a heterogeneous but related set of fit behavior tree controllers, which followed the trajectory of the island model system.

3.7. Experimental Protocol

The nine Xpucks were placed in the arena (see Figure 1) at the left hand ($x < -0.6$ m) end with random orientation. The blue frisbee was placed approximately in the center of the arena. From the Hub PC, the state of the Xpucks was monitored and the experiment was started, pausing it as necessary to relocate the frisbee back to the center of the arena. While the swarm was not paused, experiment time advanced and the evolutionary algorithm proceeded on each Xpuck. After 16 min of experiment time, the experiment was complete and the Xpucks were halted. During those 16 min, seven different controllers had run on each Xpuck.

All Vicon data, telemetry, and evolutionary algorithm data were logged for analysis. This included the heritage and measured simulation fitness of every single individual within the whole island model evolutionary system, and the full behavior trees of the fittest individuals of each island for every generation.

Because the power consumption when running the simulator for the evolutionary algorithm was high, the Xpuck battery life was about 1.5 h, sufficient for about five runs.

4. Results

We performed 20 runs. Mean final fitness of the evolutionary algorithm was 0.21, $\sigma = 0.037$. Mean fitness of the real swarm in that last 2 min segment was 0.085, $\sigma = 0.11$. The distributed island model evolutionary algorithm running on the swarm

Table 2. The 20 runs, showing f_{EA}, the mean fitness of the island model evolutionary algorithm, and f_{realx}, the real fitness for each 2 min segment of the 16 min runs. Mean and standard deviation over all runs are shown at the bottom of the table. Real fitness values $f > 0.1$ are shown in bold.

Run	f_{EA}	f_{real1}	f_{real2}	f_{real3}	f_{real4}	f_{real5}	f_{real6}	f_{real7}
1	0.219	0.001	−0.113	**0.152**	0.098	−0.004	−0.081	**0.174**
2	0.163	0.023	0.036	−0.035	0.016	0.020	0.043	0.028
3	0.220	0.048	−0.086	**0.106**	−0.024	**0.141**	**0.123**	**0.116**
4	0.233	0.034	0.017	0.038	0.042	0.039	−0.023	0.085
5	0.203	0.065	−0.063	0.040	**0.108**	**0.148**	−0.077	0.097
6	0.259	−0.025	0.030	−0.031	−0.019	0.015	**0.125**	**0.240**
7	0.266	−0.080	−0.032	**0.175**	**0.188**	**0.110**	0.073	**0.204**
8	0.213	−0.005	**0.172**	0.072	**0.126**	0.079	0.015	**0.171**
9	0.154	0.093	0.014	0.092	0.039	0.000	0.062	0.038
10	0.221	−0.043	−0.015	**0.191**	**0.132**	0.062	0.059	0.005
11	0.299	−0.004	0.043	−0.025	**0.105**	**0.114**	**0.194**	**0.200**
12	0.227	0.009	0.038	−0.096	**0.121**	0.089	**0.146**	**0.154**
13	0.180	−0.036	−0.016	−0.139	−0.001	**0.101**	−0.017	**0.199**
14	0.164	0.070	−0.009	0.074	0.097	0.085	0.028	**0.108**
15	0.167	0.054	−0.061	−0.023	**0.127**	−0.055	0.063	−0.069
16	0.184	−0.104	0.048	**0.107**	0.075	0.049	0.076	−0.082
17	0.189	0.022	−0.024	0.082	−0.069	0.098	−0.035	**0.160**
18	0.183	−0.047	0.013	0.022	0.053	0.063	0.004	**0.124**
19	0.180	−0.056	−0.026	0.038	−0.025	−0.021	−0.088	−0.197
20	0.182	0.064	0.086	0.085	0.008	0.094	−0.035	−0.061
x	0.205	0.004	0.003	0.046	0.060	0.061	0.033	0.085
σ	0.037	0.053	0.062	0.084	0.066	0.054	0.076	0.113

completed an average of 84, $\sigma = 11.1$ generations per run giving a mean generation time of 12.2 s. The swarm ran a total of 3.9 million simulations. The performance of the swarm in each run is detailed in **Table 2**, which shows the final average fitness of the island model evolutionary algorithm and the real fitness of the swarm in each 2 min segment that it was running an evolved controller. **Figure 2**a shows the mean fitness of the island model evolutionary algorithm running on the swarm for each of the 20 runs and Figure 2b shows the real swarm fitness. The evolutionary algorithm shows increasing fitness in every case, with final mean fitness over all runs of 0.21, with best and worst fitness of 0.30 and 0.15. The measured fitness of the real swarm differs from the simulated swarm fitness, but only in the case of runs 19 and 20, it differs significantly (two sample independent T-test with assumption of same variance, $p = 0.035$ and $p = 0.004$). The remaining 90% of runs have performance differences that can be attributed to sampling error.

4.1. Behavioral Analysis

One important motivation for using behavior trees is their human understandability. In this section, we classify the multiple runs to choose an analysis target. When watching videos of the real swarm, it is apparent that there is a rich variety of behaviors that solve the problem of collective movement of the frisbee, not captured by the bottom line fitness measure. The approach we take to analyze and gain insight is as follows. Firstly, we define several behavioral metrics, which we can automatically calculate from the captured trajectory data of the swarm. We then associate these metrics with individual 2 min segments during which the swarm is executing a fixed set of behavior tree controllers. In general, the segments near the end of an experimental run will have greater real fitness, but we have already seen that there is wide variance in this measure.

We take all the segments that have a reasonable real world swarm fitness, defined here to be $f > 0.1$, and shown bolded in Table 2, and perform a behavioral cluster analysis to determine the impact of quite different solution strategies. From different clusters, representing a different solution style, we can then analyze the behavior tree controllers themselves, using the identities defined earlier to simplify the trees such that we can gain understanding of their functionality. In doing this, we hope to discover useful or interesting behavioral traits.

The metrics we define are: 1) Energy $m_{\text{energy}} = \sum_{i \in \text{robots}} |v_{\text{left}}(i)| + |v_{\text{right}}(i)|$. The total use of the motors. 2) Pushing m_{push}, the average proportion of the robots that are within 1 frisbee radius plus 1.5 robot radii of the center of the frisbee. 3) Loitering m_{loit}, the average proportion of robots that are within 3 frisbee radii but not in the pushing zone. 4) Cooperation $m_{\text{coop}} = \frac{1}{n \cdot r_{\text{pushing}}} \left| \sum_{i \in \text{pushing}} (1, \angle\theta_i) \right|$, the degree to which the robots in close proximity to the frisbee are facing in the same direction, thus can push cooperatively. 5) Acceleration

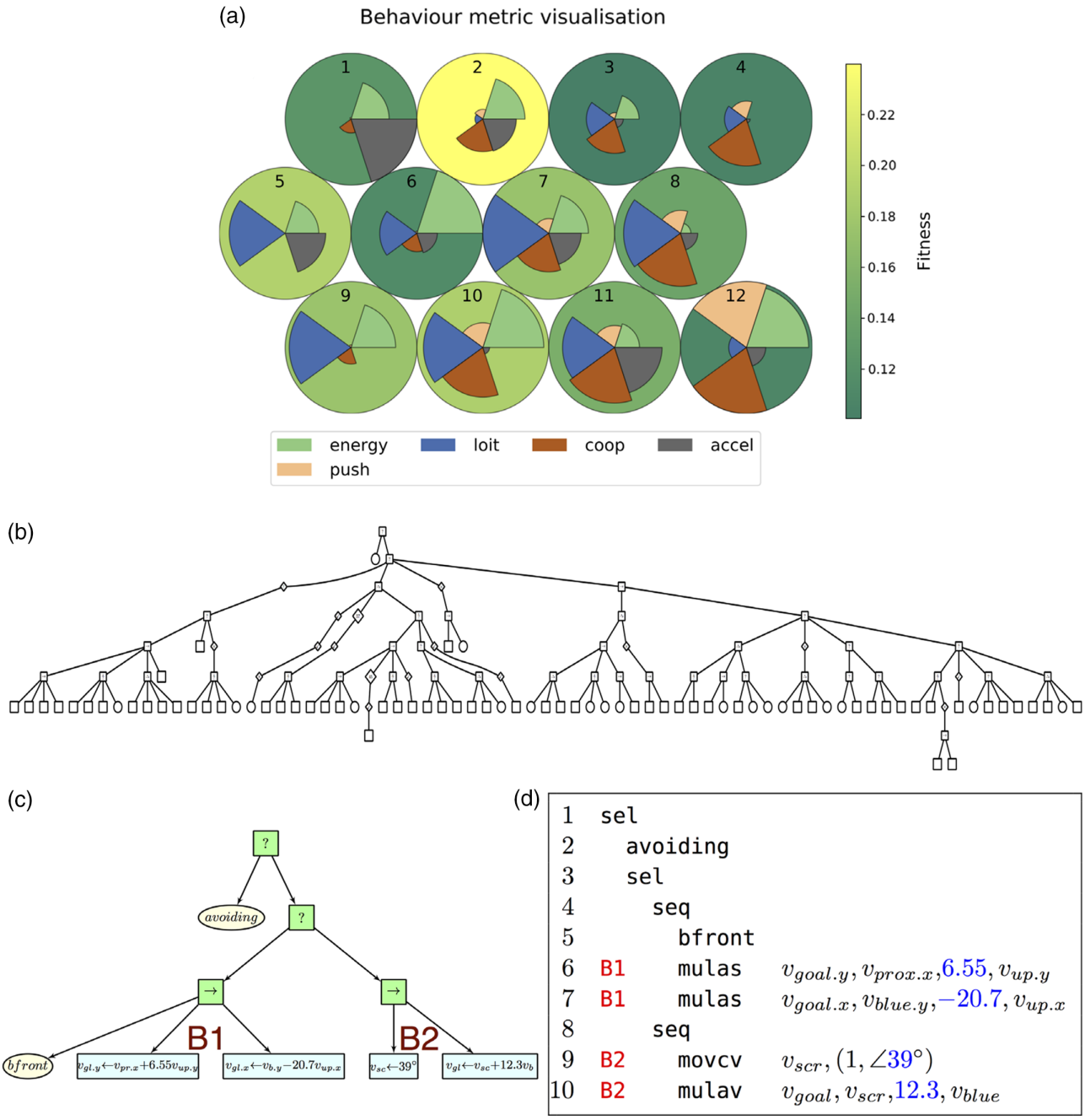

Figure 2. a) Self-organizing map clustering different real swarm run segments according to five behavior metrics m_{energy}, m_{push}, m_{loit}, m_{coop}, and m_{accel}, shown as different radii segments within cells. Background color is mean fitness of that cell. b) Outline of Tree 806768 in original evolved form showing overall structure. c,d) Tree 806768 in reduced form as a tree and listing. The two behaviors B1 and B2 are labeled in red, and parameters are highlighted in blue.

$m_{accel} = \sum |\frac{\Delta v_{mleft}}{\Delta t}| + |\frac{\Delta v_{mright}}{\Delta t}|$. The sum of all motor absolute motor accelerations, a measure of how jerky the motion is.

A total of 33 segments in the runs have a fitness $f > 0.1$. We cluster these using a self-organizing map that tries that arranges the data in cells such that topological relations in the high-dimensional feature space are somewhat preserved in the 2D representation.[40]

Figure 2a shows the map. The background color of the Cells shows the fitness, ranging from dark green in Cell 4 representing $f = 0.11$ to light yellow in cell 2, representing $f = 0.24$. The radii of the segments within the cells show the values of each of the behavioral metrics. Consider Cell 1, showing a high value for m_{accel} but low for the other metrics, indicating a predominance of quite jerky movement with few robots near the frisbee, and

Cell 12, with very high m_{energy}, m_{push}, and m_{coop}, indicating lots of movement with many robots quite close to the frisbee.

The three fittest runs in real life are Runs 6, 7, and 11, which have final segments in Cells 2, 10, and 9 respectively. Runs 7 and 11 are in adjacent cells so would be expected to have more similar behavior than to Run 6. Run 11 is interesting for another reason, that its final population of controllers is dominated by a single controller, and it was the fittest run in simulation.

For these reasons, we examine the behavior trees from Run 11 final segment, present in Cell 9. It has the highest final fitness in the EA of 0.299, and a final segment fitness in reality of 0.2, with steadily increasing real fitness over most of experimental run. The trees present in the final segment have unique identifiers 806768, 906737, 807914. Before analyzing any trees, we first perform a pairwise functional comparison between trees to eliminate those that are differently labeled but functionally identical. This shows that tree 807914 is functionally identical to 806768. There are differences in the tree, but these are in never triggered branches. Tree 906737 controls two robots but tree 806768 dominates the swarm, having migrated from its origin to be present on seven of the nine robots. This suggests that it is consistently fit.

4.2. Tree Analysis

The original and reduced versions of tree 806768 are shown in Figure 2b,c. It is clear that there is a large amount of redundancy. The original has 134 nodes and the reduced form has 9, a 93% reduction. From the reduced form, we can extract and understand the behavior, finding interesting emergent effects where two robots acting together can stably push the frisbee.

Let us analyze it, line numbers referring to the listing on the right of Figure 2f. The `sel avoiding` is the standard prefix that we use for all evolved trees to perform basic collision avoidance before any other behaviors. If the robot is not performing the avoiding action, then the subsequent tree to the right is ticked. This consists of two sequences, the first guarded by the query node `bfront`. If this returns success, if there is blue directly in front of the robot, the rest of that sequence will be ticked, otherwise the second sequence will be ticked.

There are therefore two behaviors, depending on whether the blackboard register $\boldsymbol{\nu}_{\mathbf{blue}}$ is non-zero and pointing forwards, i.e., there is something blue in the center of the field of vision of the robot. If the robot is directly facing something blue, it will perform one behavior (lines 6 and 7) labeled B1, otherwise it will perform the behavior of lines 9 and 10, labeled B2. We can restate this as

$$\nu_{goal} = \begin{cases} \begin{bmatrix} \nu_{\text{blue}.y} - 20.7 \cdot \nu_{\text{up}.x} \\ \nu_{\text{prox}.x} + 6.5 \cdot \nu_{\text{up}.y} \end{bmatrix} & \text{if directly facing frisbee (B1)} \\ (1, \angle 39^\circ) + 12.3 \cdot \nu_{\text{blue}} & \text{otherwise (B2)} \end{cases} \tag{5}$$

If not directly facing the frisbee, the behavior B2 is quite simple to state; the robot move forward in an anticlockwise circular fashion until something blue enters the visual field, at which point it move forward while turning in that direction until the frisbee is in the center of the visual field, at which point the other behavior B1 takes control. **Figure 3**a visualizes a simulation of the behavior tree with the robot starting at pose (0.3, 0, 0) so facing in the $+x$ direction away from the frisbee. The location of the robot is shown for each timestep of 100 ms over a period of 3 s. The color of the trail indicates which behavior is executing; yellow indicating B2 and green B1. We can see that the robot circles in an anticlockwise direction until the blue frisbee comes into view, at which point the robot heads more toward the frisbee. Finally the behavior switches to B1.

Behavior B1, triggered when the robot is directly facing the frisbee, forms the goal vector of several components and the meaning is not immediately clear. By observing that the components of the ν_{up} vector dominate the maximum values that might be seen from $\boldsymbol{\nu}_{\mathbf{prox}.x}$ and $\boldsymbol{\nu}_{\mathbf{blue}.y}$ of ≈ 2 and 0.32 respectively, the majority of $\boldsymbol{\nu}_{\mathbf{goal}}$ is formed from the $\boldsymbol{\nu}_{\mathbf{up}}$ vector reflected in the robot y-axis and anisotropically scaled. If the pose angle of the robot is between $[-\pi/2, \pi/2]$, this will cause the robot to rotate to face in the $+x$ direction and stop. With angles greater than this, $[>\pi/2, <-\pi/2]$, i.e., facing in the $-x$ direction, the robot will move forward while turning to face the $+x$ direction. The rate of turning is dependent on the angle of the robot, so at an angle of exactly π, the robot will move forward with no turning, but any deviation will result in an accelerating turn toward the $+x$ direction.

Consider two scenarios, each with the frisbee in the center and one with a robot to its right facing in the $-x$ direction, and one with a robot to its left facing in the $-x$ direction. In the first scenario, the robot will move forward until it contacts the frisbee, then pushing the frisbee in the $-x$ direction. In the second scenario, the robot will not move. We can see that random creation of these two scenarios will on average result in an increase in fitness because the frisbee will only ever move in the $-$x direction. If we perturb the first scenario slightly with a robot starting pose of $(0.2, 0, \pi - 0.1)$, the robot will move forward while turning clockwise. The frisbee will again be pushed in the $-x$ direction, but as the robot continues to rotate, it will reach the situation where the vector $\boldsymbol{\nu}_{\mathbf{blue}}$ no longer has zero angle (from the possible angles of -18.7°, -9.35°, 0, 9.35°, 18.7°) and thus `bfront` will return failure and behavior B2 will occur. This will tend to make the robot move forward and turn anticlockwise toward the frisbee, while pushing it. If the turning rate is fast enough, then the robot will end up fully facing the frisbee again, such that the first behavior is again activated. We can see that we might have a switching of behaviors of clockwise and anticlockwise forward movement such that the frisbee is on average moved in the $-x$ direction.

In fact, a single robot does not reliably turn far enough that the frisbee becomes centered in the field of view, so sometimes, the frisbee will get pushed in a circular path and sometimes on an erratic path toward $-$x depending on the exact starting condition. Figure 3b shows the evolution of the perturbed first scenario, with B1 resulting in a slow clockwise turn initially, then a period of rapid switching between B1 and B2, a further period of B1 turning anticlockwise this time, then finally a stable situation running B2 with the robot pushing the frisbee in a circular path.

What is interesting is if we change the scenario to have two robots in contact with the frisbee. In this case, although neither individually can stably push the frisbee, with two robots their interactions produce an emergent stable pushing behavior. It is important to realize that these interactions now include

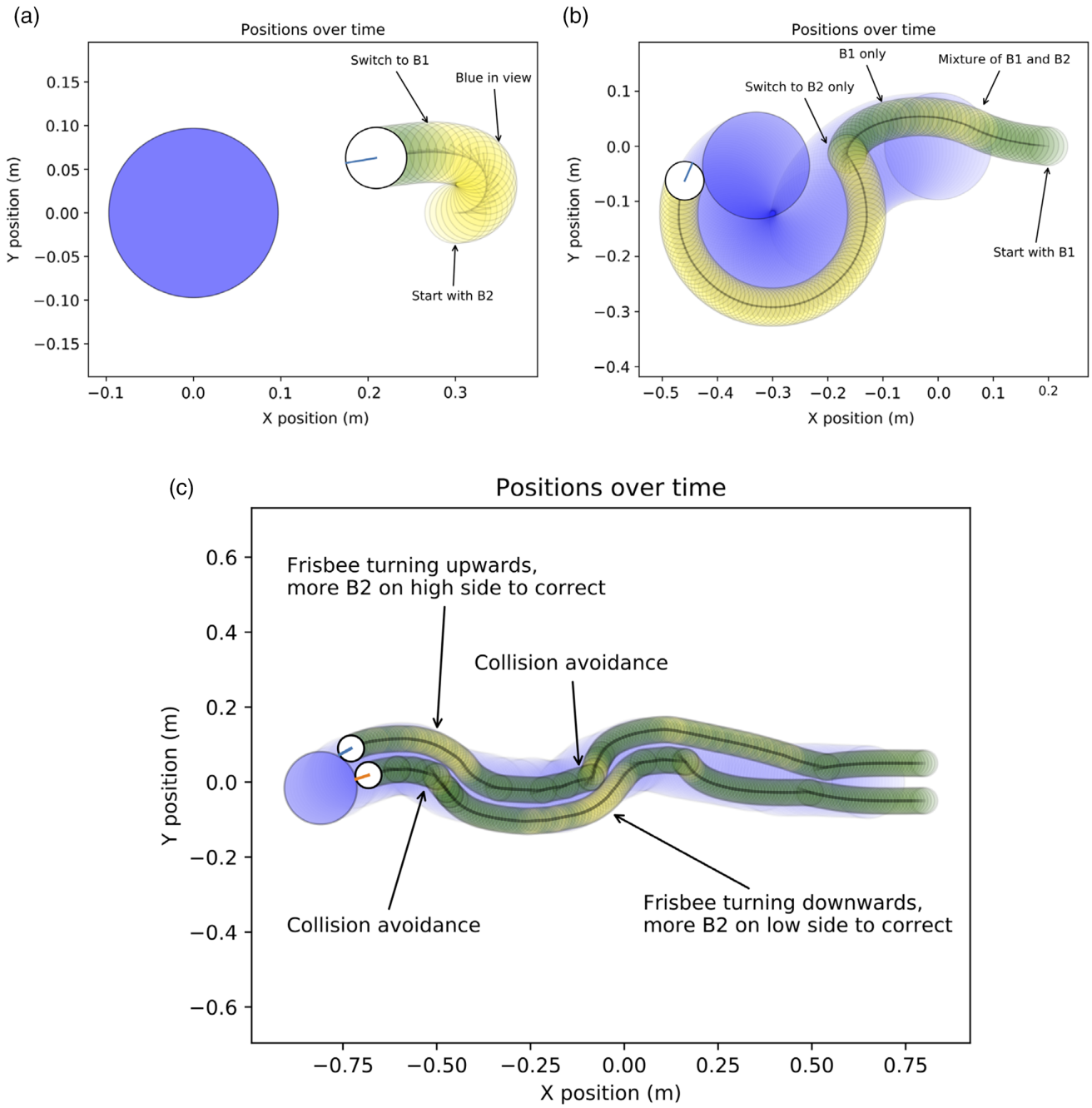

Figure 3. Visualizations of various scenarios. a) A single Xpuck following behavior B2 then B1. The robot starting pose is (0.3,0,0), facing in the +x direction away from the frisbee. Color of the trail is green for B1 and yellow for B2, each plot of trail is one control cycle of 100 ms. b) Single Xpuck following behavior B1 at the start, then combinations until ending in a stable orbit in behavior B2. c) Two Xpucks with a starting position close to the right of the frisbee exhibiting stable emergent cooperative pushing behavior.

the default collision avoidance behavior, not shown in Equation (5). This usually causes a robot to turn on the spot away from the object detected with the IR proximity sensors and is visible on the trail visualization as denser outlines at points where the robots are not moving forward.

Figure 3c shows an example of this. The initial configuration of the system was the frisbee at location (0.65, 0) and the robots at poses (0.8, 0.05, π) and (0.8, -0.05, π). The track of the frisbee is not straight, but never degenerates into a stable orbit. The two robots use varying amounts of B1 and B2, depending on the system state. By inspection, when the frisbee path is tending upward too much, the top Xpuck starts spending more time in B2 and the less in B1, causing the system of frisbee and robots to turn back downward, and likewise in the opposite situation, with collision

avoidance ensuring that the other robot is turned to maintain some separation.

4.2.1. Resilience to Perturbation

The two robot scenario demonstrating emergent pushing above is ideal, in that the starting positions are adjacent to the frisbee and facing in the correct direction. How resilient is this system to perturbations of the positions of the two robots to the relative to the frisbee? We approach this by comparing the performance of the tree 806768 with the simple tree that just moves forward and performs collision avoidance, called forward. We start with the frisbee at position (0, 0) and the two robots are placed with random poses centered on (0, 0, 0) with added Gaussian noise of standard deviations (0.2, 0.2, 1.5). The robots and frisbee are not allowed to overlap. Valid configurations are simulated for a time of 8 s, just less than the time for a perfect attempt to push the frisbee to an x-boundary. A total of 100 000 simulations were run for each tree. The mean starting distance of the robots from the frisbee is measured for each run, and this data are binned and plotted against the fitness of that run.

Figure 4a shows the results. There are clearly two quite different types of behavior here. As you might expect, if you run a lot of trials of essentially a random walk (move forward with collision

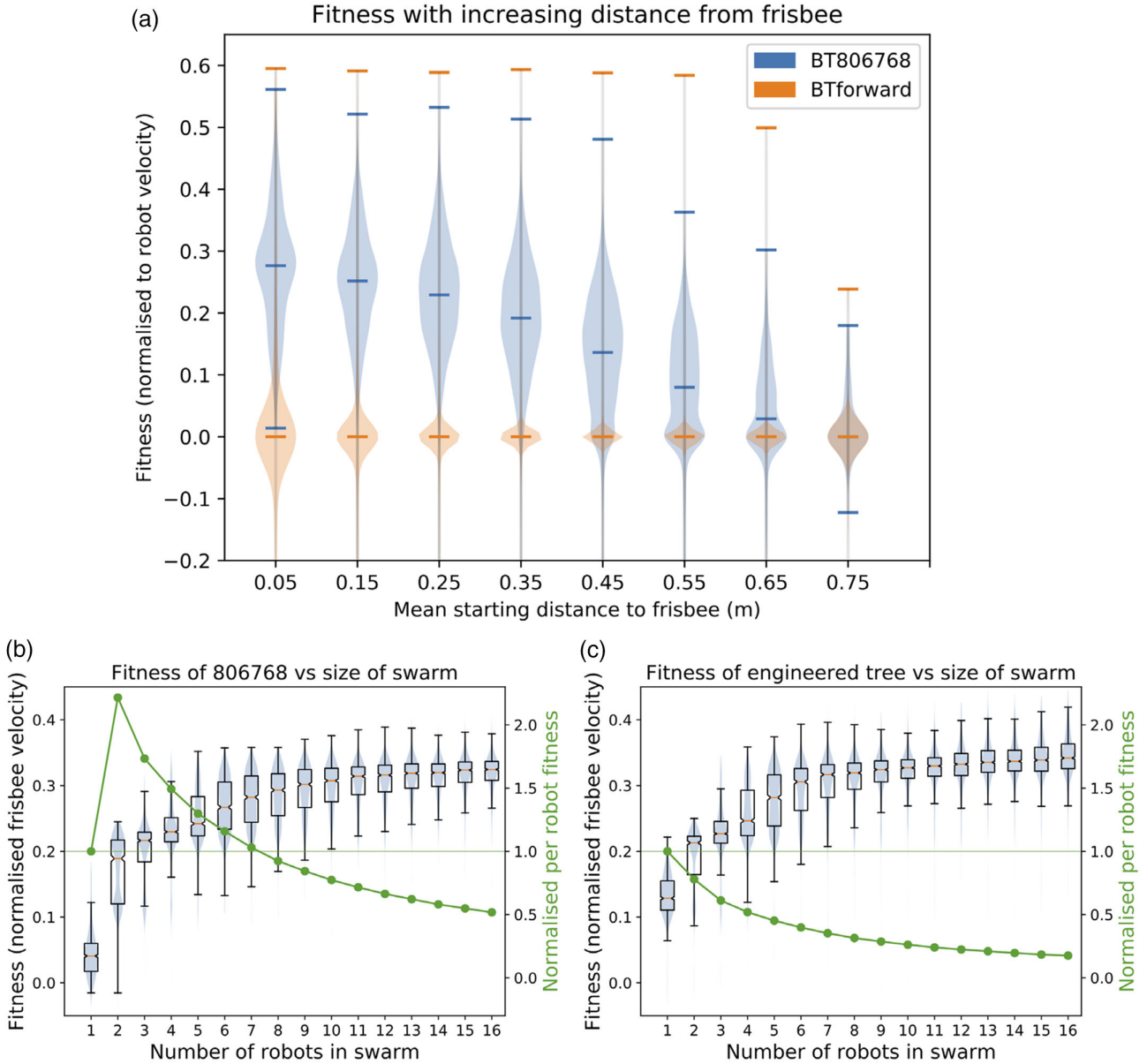

Figure 4. a) Distribution of fitness of two robots over 100 000 simulations of each of two trees against mean starting distance from frisbee. Ticks show extrema and medians. The tree 806768 maintains good performance even when the starting position is far from the frisbee, indicating ability to find and then push the frisbee. b) Performance of tree 806768 with varying swarm size. The swarm exhibits superlinear performance scaling up to a swarm size of seven. c) Performance of engineered version of tree 806768 showing better single-robot performance and 10% better overall performance.

avoidance), there will be some that are quite fit, but the majority will not be. The data show this, with the forward tree having most runs clustering around zero fitness. The tree 806768, in contrast, maintains a consistent median fitness, which falls gradually as the mean starting distance from the frisbee increases, as you would expect because the robots have to reach the frisbee before pushing it. The important indicator of an effective controller is that the median fitness is maintained even over a quite large increase in the distance away from the frisbee, implying the active movement toward and then pushing of the frisbee.

4.2.2. Scalability

One interesting question about swarm controllers is whether they produce emergent behavior. It is not obvious how to answer this, but one approach would be to measure the fitness of the controller when running in different sized swarms. If there was no emergent behavior, we would expect a single agent to have a certain degree of fitness $f = f_{\text{agent}}$, then n agents to have a higher fitness $f = k \cdot f_{\text{agent}}$ but with $k < n$ since multiple agents with no cooperation or emergent behavior may interfere, and for our task there is a physical limit on how many agents can actually interact with the frisbee. We expect sublinear scaling, in other words. Conversely, with emergent cooperation in the swarm, we may see superlinear scaling when cooperation outweighs interference. Superlinear performance scaling has been observed in swarm robotics systems,[41] and Hamann develops a simple model of swarm performance comprising two components of cooperation and interference.[42]

We simulated the tree 806768 at different swarm sizes up to $n = 16$. Figure 3b shows the results. There is clearly superlinear performance scaling up to a swarm size of $n = 7$. Above seven robots, the performance scaling is sublinear as the system performance reaches a plateau of around $f = 0.3$. Above a certain number of robots, there can be no improvement in performance because there is only a single frisbee and the robots have a maximum velocity. We can regard the superlinear scaling as evidence of emergent collective behavior.

4.3. Engineering Higher Performance

Given the ability we now have to deconstruct evolved behavior trees and understand how they work; we can use this knowledge to engineer higher performance. We observed in Section 4.2 that a single robot using tree 806768 was unable to reliably push the frisbee, while more than one robot could. What if we could tune the behavior tree such that a single robot could successfully push the frisbee? The listing in Figure 2f shows four parameters highlighted in blue. We denote these *a*, *b*, *c*, and *d*, respectively. We decided to try and hand tune the parameters to optimize both single-robot pushing stability and overall fitness.

We can see from the data in **Table 3** that we have achieved a useful and significant (independent T-test $p < 0.0001$) performance improvement of about 10% from what was already a quite fit controller. The performance was measured in simulation over 1000 runs with different starting conditions before and after tuning. Figure 4c shows that the single agent performance has increased considerably from the unaltered tree (from 0.039 to 0.12) as we intended when optimizing for more stable single-robot pushing behavior.

Table 3. Hand-tuned parameters for tree 806768, performance measured in simulation over 1000 runs with different starting conditions.

	Original	Optimized
A	6.55	2
B	−20.7	−48
C	39°	70.3°
D	12.3	48
Fitness $\bar{x}$	0.27	0.30
Fitness σ	0.044	0.041

5. Conclusions

We have demonstrated a swarm that is capable of evolving new controllers within the swarm itself, removing the tie to off-line processing power. The in-swarm computational power is able to run an island model evolutionary algorithm that can produce fit and effective swarm controllers within 15 real-time minutes, far faster than has been possible previously. This is due to careful attention to several elements; the writing of a fast simulator that makes maximal use of the GPU processing power available, tuning the simulator parameters and controller architecture to minimize and mitigate reality gap effects, using the available simulator budget more effectively by improving the evolutionary algorithm, and finally using the island model to scale the evolutionary performance with the size of the swarm. The progressive transfer of control of the real robots to these better controllers leads to improving real-world fitness.

One overarching theme of this study has been the desirable properties of behavior trees as a controller architecture, particularly as the target of evolutionary algorithms. The modularity, human understandability, and natural extendibility mean that we can analyze and understand evolved controllers for insight. In this study, we demonstrate this by using automatic methods to simplify evolved trees, then further human analysis to describe in detail how a selected tree actually functions. We then demonstrate how this confers control to the human who can even improve the performance of the swarm. This understandability, or explainability, is an important characteristic for the safety of future systems created by machine learning.

This study opens the door to the automatic design of robot swarm behaviors in the wild, while providing a human-understandable interface that can be queried and modified by a human operator. We believe this will be the first step toward deploying robots in real-world applications in a fully automatic and adaptable way.

Supporting Information

Supporting Information is available from the Wiley Online Library or from the author.

Acknowledgements

S.J. was funded by EPSRC grant EP/L015293/1.

Conflict of Interest

The authors declare no conflict of interest.

Keywords

evolutionary robotics, swarm robotics, behavior trees, explainability

Received: May 27, 2019
Revised: July 15, 2019
Published online:

[1] E. Şahin, *Swarm Robotics*, Springer, Berlin **2005**, p. 10.
[2] M. Brambilla, E. Ferrante, M. Birattari, M. Dorigo, *Swarm Intell.* **2013**, *7*, 1.
[3] S. Hauert, S. Mitri, L. Keller, D. Floreano, in *The Horizons of Evolutionary Robotics* (Eds: P. A. Vargas, E. A. Di Paolo, I. Harvey, P. Husbands), MIT Press, Cambridge, MA **2014**, pp. 203–217.
[4] S. Jones, M. Studley, S. Hauert, A. Winfield, *Front. Robot. AI* **2018**, *5*, 11.
[5] C. W. Reynolds, *ACM SIGGRAPH Comput. Graph.* **1987**, *21*, 25.
[6] V. Trianni, R. Groß, T. H. Labella, E. Şahin, M. Dorigo, *Adv. Artif. Life* **2003**, *1*, 865.
[7] S. Hauert, J.-C. Zufferey, D. Floreano, in Proc. IEEE Congress on Evolutionary Computation, IEEE, Piscataway, NJ **2009**, p. 55.
[8] V. Trianni, S. Nolfi, *Artif. Life* **2011**, *17*, 183.
[9] M. Rubenstein, A. Cornejo, R. Nagpal, *Science* **2014**, *345*, 795.
[10] L. Pitonakova, R. Crowder, S. Bullock, *Front. Robot. AI* **2018**, *5*, 47.
[11] G. Baldassarre, S. Nolfi, D. Parisi, *Artif. Life* **2003**, *9*, 255.
[12] G. Francesca, M. Brambilla, A. Brutschy, L. Garattoni, R. Miletitch, G. Podevijn, A. Reina, T. Soleymani, M. Salvaro, C. Pinciroli, F. Mascia, V. Trianni, M. Birattari. *Swarm Intell.* **2015**, *9*, 125.
[13] M. Duarte, J. Gomes, V. Costa, S. M. Oliveira, A. L. Christensen, *Applications of Evolutionary Computation* (Eds: G. Squillero, P. Burelli), Springer, Cham, **2016**, p. 213.
[14] S. Jones, M. Studley, S. Hauert, A. Winfield, in Proc. 13th Int. Symp. on Distributed Autonomous Robotic Systems (Eds: R. Gross, A. Kolling, S. Berman, E. Frazzoli, A. Martinoli, F. Matsuno, M. Gauci), Springer, Cham **2018**, p. 487.
[15] G. Francesca, M. Birattari, *Front. Robot. AI* **2016**, *3*, 29.
[16] P. Ogren, in Proc. AIAA Guidance, Navigation and Control Conf., American Institute of Aeronautics and Astronautics, Reston, VA **2012**, p. 4458.
[17] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, Vol. 1, MIT Press, Cambridge, MA **1992**.
[18] M. Colledanchise, P. Ogren, in Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (Ed: W. Burgard), IEEE, Piscataway, NJ **2014**, p. 1482.
[19] A. Marzinotto, M. Colledanchise, C. Smith, P. Ogren, in Proc. IEEE Int. Conf. on Robotics and Automation (Ed: W. Burgard), IEEE, Piscataway, NJ **2014**, p. 5420.
[20] K. Y. Scheper, S. Tijmons, C. C. de Visser, G. C. de Croon, *Artif. Life* **2016**, *22*, 23.
[21] S. Mitri, D. Floreano, L. Keller, *Proc. R. Soc. B Biol. Sci.* **2010**, *278*, 378.
[22] M. Waibel, L. Keller, D. Floreano, *IEEE Trans. Evol. Comput.* **2009**, *13*, 648.
[23] N. Jakobi, P. Husbands, I. Harvey, in Proc. European Conf. on Artificial Life (Eds: F. Morán, A. Moreno, J. J. Merelo, P. Chacón), Springer, Berlin **1995**, p. 704.
[24] S. Koos, J.-B. Mouret, S. Doncieux, *IEEE Tran. Evol. Comput*. **2013**, *17*, 122.
[25] J.-B. Mouret, K. Chatzilygeroudis, in Proc. Genetic and Evolutionary Computation Conf. (Ed: G. Ochoa), ACM, New York, NY **2017**, p. 1121.
[26] R. A. Watson, S. G. Ficici, J. B. Pollack, *Robot. Auton. Syst.* **2002**, *39*, 1.
[27] Y. U. Takaya, T. Arita, presented at 8th Int. Symp. on Artificial Life and Robotics, Beppu, Japan, January **2003**.
[28] N. Bredeche, J.-M. Montanier, W. Liu, A. F. Winfield, *Math. Comput. Model. Dyn. Syst.* **2012**, *18*, 101.
[29] S. Doncieux, N. Bredeche, J.-B. Mouret, A. E. G. Eiben, *Front. Robot. AI*, **2015**, *2*, 4.
[30] N. Bredeche, E. Haasdijk, A. Prieto, *Front. Robot. AI* **2018**, *5*, 12.
[31] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, A. Martinoli, in Proc. 9th Conf. on Autonomous Robot Systems and Competitions, (Eds: P. J. S. Gonçalves, P. Torres, C. M. O. Alves), Vol. 1, IPCB, Portugal **2009**, p. 59.
[32] E. Cantú-Paz, *Calculateurs paralleles, reseaux et systems repartis* **1998**, *10*, 141.
[33] D. Whitley, S. Rana, R. B. Heckendorn, *CIT. J. Comp. Inform. Technol.* **1999**, *7*, 33.
[34] P. J. O'Dowd, M. Studley, A. F. Winfield, *Evol. Intel.* **2014**, *7*, 95.
[35] W. Liu, A. F. Winfield, *Microprocess. Microsy.* **2011**, *35*, 60.
[36] A. F. Winfield, *Encyclopedia of Complexity and Systems Science* (Ed: R. A. Meyers), Springer, New York **2009**, 3682.
[37] G. Francesca, M. Brambilla, A. Brutschy, V. Trianni, M. Birattari, *Swarm Intel.* **2014**, *8*, 89.
[38] D. Whitley, S. Rana, R. B. Heckendorn, in Proc. AISB International Workshop on Evolutionary Computing, Springer, Berlin, **1997**, p. 109.
[39] R. Poli, W. B. Langdon, N. F. McPhee, J. R. Koza, A Field Guide to Genetic Programming. http://lulu.com; http://www.gp-field-guide.org.uk (accessed: December 2008).
[40] T. Kohonen, *Biol. Cybern.* **1982**, *43*, 59.
[41] F. Mondada, M. Bonani, A. Guignard, S. Magnenat, C. Studer, D. Floreano, in Proc. European Conf. on Artificial Life (Eds: M. S. Capcarrère, A. A. Freitas, P. J. Bentley, C. G. Johnson, J. Timmis), Springer, Berlin **2005**, p. 282.
[42] H. Hamann, in Proc. Int. Conf. on Swarm Intelligence (Eds: Y. Tan, Y. Shi, Z. Ji), Springer, Berlin **2012**, p. 168.

